Lab Cycle 4

1. Using the iris data set implement the KNN algorithm. Take different values for Test and training data set. Also use different values for k. Also find the accuracy level.

Input:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
print("Name: Athul Ajay")
print("Reg No: SJC22MCA-2017")
print("Batch: 22-24")
print()
iris_data = pd.read_csv('iris.csv')

X = iris_data.iloc[:, :-1].values
y = iris_data.iloc[:, -1].values

test_sizes = [0.5, 0.2, 0.7]
k_values = [1,5,9]

for test_size in test_sizes:
    for k in k_values:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)

        knn = KNeighborsClassifier(n_neighbors=k)

        knn.fit(X_train, y_train)

        y_pred = knn.predict(X_test)

        accuracy = accuracy_score(y_test, y_pred)

        print(f'Test Size: {test_size}, k: {k}, Accuracy: {accuracy}')
```

Output:

```
/home/sjcet/PycharmProjects/Athul/venv/bin/python /home/sjcet/Pyc
Name: Athul Ajay
Reg No: SJC22MCA-2017
Batch: 22-24

Test Size: 0.5, k: 1, Accuracy: 0.9466666666666667
Test Size: 0.5, k: 5, Accuracy: 0.9333333333333333
Test Size: 0.5, k: 9, Accuracy: 0.9866666666666667
Test Size: 0.2, k: 1, Accuracy: 1.0
Test Size: 0.2, k: 5, Accuracy: 0.9333333333333333
Test Size: 0.2, k: 9, Accuracy: 0.9
Test Size: 0.7, k: 1, Accuracy: 0.9333333333333333
Test Size: 0.7, k: 5, Accuracy: 0.9523809523809523
Test Size: 0.7, k: 9, Accuracy: 0.9333333333333333

Process finished with exit code 0
```

2. Download another data set suitable for the KNN and implement the KNN algorithm. Take different values for Test and training data set. Also use different values for k.

Input:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
print("Name: Athul Ajay")
print("Reg No: SJC22MCA-2017")
print("Batch: 22-24")
print()
data = pd.read_csv('diabetes.csv')

X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

test_sizes = [0.5, 0.2, 0.7]
k_values = [1,3,7]

for test_size in test_sizes:
    for k in k_values:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)

        # Create a KNN classifier
        knn = KNeighborsClassifier(n_neighbors=k)

        # Train the model
        knn.fit(X_train, y_train)
```

```
    # Make predictions
    y_pred = knn.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    print(f'Test Size: {test_sizes}, k: {k}, Accuracy: {accuracy}')
```

Output:

```
/home/sjcet/PycharmProjects/Athul/venv/bin/python /home/sjcet/Pycl
Name: Athul Ajay
Reg No: SJC22MCA-2017
Batch: 22-24

Test Size: [0.5, 0.2, 0.7], k: 1, Accuracy: 0.6666666666666666
Test Size: [0.5, 0.2, 0.7], k: 3, Accuracy: 0.6614583333333334
Test Size: [0.5, 0.2, 0.7], k: 7, Accuracy: 0.7317708333333334
Test Size: [0.5, 0.2, 0.7], k: 1, Accuracy: 0.7012987012987013
Test Size: [0.5, 0.2, 0.7], k: 3, Accuracy: 0.6818181818181818
Test Size: [0.5, 0.2, 0.7], k: 7, Accuracy: 0.7857142857142857
Test Size: [0.5, 0.2, 0.7], k: 1, Accuracy: 0.6728624535315985
Test Size: [0.5, 0.2, 0.7], k: 3, Accuracy: 0.7174721189591078
Test Size: [0.5, 0.2, 0.7], k: 7, Accuracy: 0.6933085501858736

Process finished with exit code 0
```

3. Using iris data set, implement naive bayes classification for different naive Bayes classification algorithms ((i) gaussian (ii) bernoulli, etc)

 Find out the accuracy level w.r.t to each algorithm

 Display the no: of mislabeled classification from test data set

 List out the class labels of the mismatching records

Input:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
print("Name: Athul Ajay")
print("Reg No: SJC22MCA-2017")
print("Batch: 22-24")
print()
dataset = pd.read_csv('iris.csv')
X = dataset.iloc[:,:4].values
y = dataset['variety'].values
dataset.head (5)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5)
from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB ()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(y_pred)

from sklearn.metrics import confusion_matrix
cm =confusion_matrix(y_test, y_pred)
print(cm)

from sklearn.metrics import accuracy_score
print ("Accuracy: ", accuracy_score (y_test, y_pred))
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
print(df)
print()
from sklearn.naive_bayes import BernoulliNB
classif = BernoulliNB ()
classif.fit(X_train, y_train)
y_pred = classif.predict(X_test)
print(y_pred)
```

```
from sklearn.metrics import confusion_matrix
cmx =confusion_matrix(y_test, y_pred)
print(cmx)

from sklearn.metrics import accuracy_score
print ("Accuracy: ", accuracy_score (y_test, y_pred))
fd = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
print(fd)
```

Output:

```
Name: Athul Ajay
Reg No: SJC22MCA-2017
Batch: 22-24

['Versicolor' 'Virginica' 'Virginica' 'Virginica' 'Setosa' 'Versicolor'
 'Virginica' 'Versicolor' 'Setosa' 'Versicolor' 'Setosa' 'Virginica'
 'Versicolor' 'Setosa' 'Setosa' 'Setosa' 'Virginica' 'Versicolor' 'Setosa'
 'Setosa' 'Virginica' 'Virginica' 'Versicolor' 'Virginica' 'Versicolor'
 'Versicolor' 'Versicolor' 'Virginica' 'Setosa' 'Setosa' 'Virginica'
 'Setosa' 'Versicolor' 'Versicolor' 'Versicolor' 'Setosa' 'Setosa'
 'Setosa' 'Virginica' 'Setosa' 'Versicolor' 'Setosa' 'Setosa' 'Virginica'
 'Virginica' 'Setosa' 'Versicolor' 'Setosa' 'Virginica' 'Virginica'
 'Versicolor' 'Virginica' 'Virginica' 'Setosa' 'Setosa' 'Virginica'
 'Versicolor' 'Setosa' 'Virginica' 'Versicolor' 'Setosa' 'Virginica'
 'Versicolor' 'Setosa' 'Virginica' 'Versicolor' 'Setosa' 'Virginica'
 'Versicolor' 'Versicolor' 'Setosa' 'Setosa' 'Versicolor' 'Versicolor'
 'Setosa']
[[28  0  0]
 [ 0 24  0]
 [ 0  0 23]]
Accuracy:  1.0
   Real Values Predicted Values
0   Versicolor       Versicolor
1    Virginica        Virginica
2    Virginica        Virginica
3    Virginica        Virginica
4       Setosa           Setosa
..         ...              ...
70      Setosa           Setosa
71      Setosa           Setosa
```

```
70      Setosa           Setosa
71      Setosa           Setosa
72  Versicolor       Versicolor
73  Versicolor       Versicolor
74      Setosa           Setosa

[75 rows x 2 columns]

['Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Virginica' 'Virginica']
[[ 0  0 28]
 [ 0  0 24]
 [ 0  0 23]]
Accuracy:  0.30666666666666664
```

```
   Real Values Predicted Values
0   Versicolor        Virginica
1    Virginica        Virginica
2    Virginica        Virginica
3    Virginica        Virginica
4       Setosa        Virginica
..         ...              ...
70      Setosa        Virginica
71      Setosa        Virginica
72  Versicolor        Virginica
73  Versicolor        Virginica
74      Setosa        Virginica

[75 rows x 2 columns]
```

4. Use car details CSV file and implement decision tree algorithm

 Find out the accuracy level.

 Display the no: of mislabeled classification from test data set

 List out the class labels of the mismatching records

Input:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
print("Name: Athul Ajay")
print("Reg No: SJC22MCA-2017")
print("Batch: 22-24")
print()
data = pd.read_csv('car.csv')
print(data.head())
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
data.columns = col_names
print(col_names)

data['class'],class_names = pd.factorize(data['class'])
data['buying'],_ = pd.factorize(data['buying'])
data['maint'],_ = pd.factorize(data['maint'])
data['doors'],_ = pd.factorize(data['doors'])
data['persons'],_ = pd.factorize(data['persons'])
data['lug_boot'],_ = pd.factorize(data['lug_boot'])
data['safety'],_ = pd.factorize(data['safety'])

print(data.head())
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

tree1 = DecisionTreeClassifier()

tree1.fit(X_train, y_train)
y_pred = tree1.predict(X_test)
# how did our model perform?
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples count:', count_misclassified)

accuracy = accuracy_score (y_test, y_pred)
print("Accuracy:", accuracy)
```

Output:

```
Name: Athul Ajay
Reg No: SJC22MCA-2017
Batch: 22-24

   vhigh vhigh.1  2 2.1  small   low  unacc
0  vhigh   vhigh  2   2  small   med  unacc
1  vhigh   vhigh  2   2  small  high  unacc
2  vhigh   vhigh  2   2    med   low  unacc
3  vhigh   vhigh  2   2    med   med  unacc
4  vhigh   vhigh  2   2    med  high  unacc
['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
   buying  maint  doors  persons  lug_boot  safety  class
0       0      0      0        0         0       0      0
1       0      0      0        0         0       1      0
2       0      0      0        0         1       2      0
3       0      0      0        0         1       0      0
4       0      0      0        0         1       1      0
Misclassified samples count: 12
Accuracy: 0.976878612716763
```

5. Implement Simple linear regression for the data set ‘student_score’

Input:

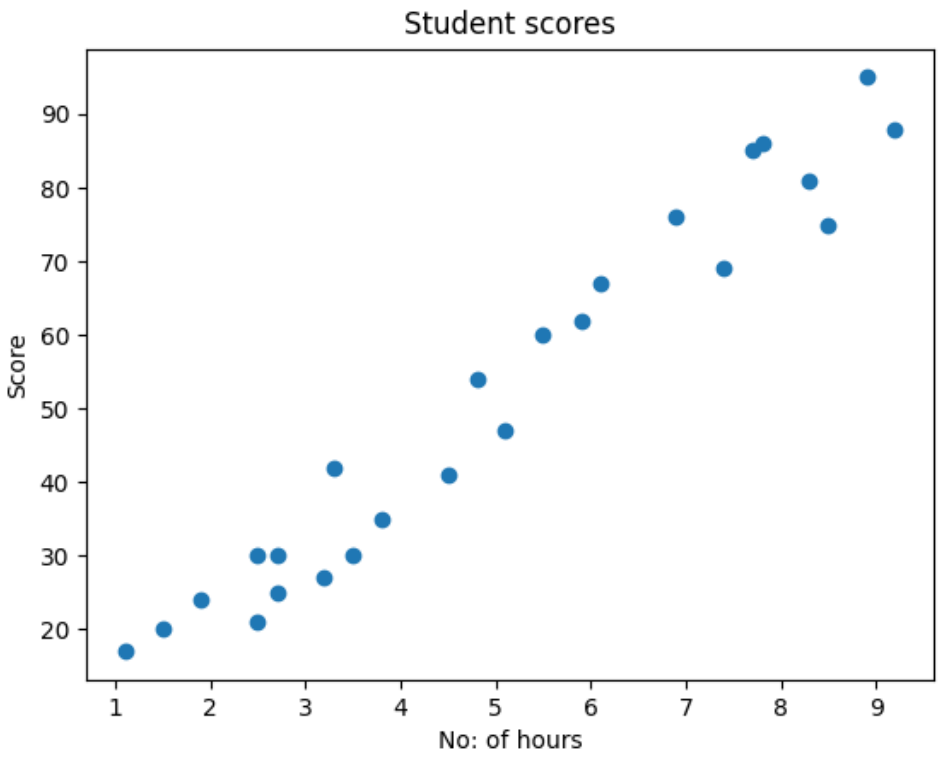
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
stud = pd.read_csv('student_scores.csv')
stud.describe()
stud.info()

Xax = stud.iloc[:,0]
Yax = stud.iloc[:,1]
plt.scatter(Xax, Yax)
plt.xlabel("No: of hours")
plt.ylabel("Score")
plt.title("Student scores")
plt.show()
X = stud.iloc[:,:-1]
y = stud.iloc[:, 1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
#print(X_train)
print("Name: Athul Ajay")
print("Reg No: SJC22MCA-2017")
print("Batch: 22-24")
print()
reg = LinearRegression()
reg.fit(X_train, y_train)
print('Intercept: ', reg.intercept_)
```

```
print('Co Efficient: ', reg.coef_)
y_pred = reg.predict(X_test)
for(i,j) in zip(y_test, y_pred):
    if(i!=j):
        print('Actual value: ', i, 'Predicted value: ',j)
print('No: of mislabeled points: ', (y_test != y_pred).sum())
print("Mean Absolute error :", metrics.mean_absolute_error(y_test,y_pred))
print("Mean Squared error :", metrics.mean_squared_error(y_test,y_pred))
print("Root Mean Squared error :",
np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

Output:



```
/home/sjcet/PycharmProjects/Athul/venv/bin/python /home/sjcet
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Hours   25 non-null     float64
 1   Scores  25 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 528.0 bytes
Name: Athul Ajay
Reg No: SJC22MCA-2017
Batch: 22-24

Intercept:  2.8075830585848323
Co Efficient:  [9.69946741]
Actual value:  47 Predicted value:  52.27486685265027
Actual value:  42 Predicted value:  34.81582551356835
Actual value:  95 Predicted value:  89.13284301293433
Actual value:  81 Predicted value:  83.31316256657368
Actual value:  25 Predicted value:  28.996145067207713
No: of mislabeled points:  5
Mean Absolute error : 4.9271011919857965
Mean Squared error : 27.036002186624764
Root Mean Squared error : 5.19961558065832

Process finished with exit code 0
```

6. Implement Multiple linear regression for the data set ‘company_data’

Input:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

advertising = pd.read_csv('Company_data.csv')
advertising.head()
advertising.describe()
advertising.info()

X = advertising.iloc[:, :-1]
y = advertising.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
reg = LinearRegression()
reg.fit(X_train, y_train)
print("Name: Athul Ajay")
print("Reg No: SJC22MCA-2017")
print("Batch: 22-24")
print()
print('Intercept is:', reg.intercept_)
print('Co Efficients are:', reg.coef_)
y_pred = reg.predict(X_test)

for(i,j) in zip(y_test, y_pred):
    if(i!=j):
        print('Actual value: ', i, 'Predicted value: ', j)
print('No: of mislabeled points: ', (y_test != y_pred).sum())

print("Mean Absolute error :", metrics.mean_absolute_error(y_test,y_pred))
print("Mean Squared error :", metrics.mean_squared_error(y_test,y_pred))
print("Root Mean Squared error :",
np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
Name: Athul Ajay
Reg No: SJC22MCA-2017
Batch: 22-24

Intercept is: 4.683499685760591
Co Efficients are: [0.05301476 0.11545787 0.00540542]
Actual value:  15.0 Predicted value:  15.710332208077403
Actual value:  16.4 Predicted value:  16.0842790240091
Actual value:  16.6 Predicted value:  16.920228490291155
Actual value:  13.3 Predicted value:  13.626743402265404
Actual value:  7.0 Predicted value:  8.458046248525532
Actual value:  12.0 Predicted value:  11.99010375027427
Actual value:  20.7 Predicted value:  19.30709580778792
Actual value:  11.0 Predicted value:  11.847102255048785
Actual value:  8.5 Predicted value:  9.215492174367322
Actual value:  25.4 Predicted value:  23.917219806901254
Actual value:  5.6 Predicted value:  7.487183634291457
Actual value:  17.1 Predicted value:  19.13903624484444
Actual value:  11.3 Predicted value:  9.420140902499412
```

```
Actual value:  8.7 Predicted value:  11.108202889243756
Actual value:  7.2 Predicted value:  11.196024701339324
Actual value:  8.1 Predicted value:  7.866380941159058
Actual value:  1.6 Predicted value:  9.339768833567078
Actual value:  5.3 Predicted value:  5.562554998521248
Actual value:  19.7 Predicted value:  20.980757809666844
Actual value:  19.4 Predicted value:  20.41157068396292
Actual value:  25.5 Predicted value:  24.925554035846602
Actual value:  10.3 Predicted value:  12.389488694789328
Actual value:  9.7 Predicted value:  9.461647354138012
Actual value:  16.0 Predicted value:  18.552389265886397
Actual value:  15.2 Predicted value:  15.271485666097066
Actual value:  10.5 Predicted value:  10.494208669232094
Actual value:  10.9 Predicted value:  10.607296514618227
Actual value:  22.6 Predicted value:  21.1845407972141
Actual value:  16.9 Predicted value:  17.966396662562325
Actual value:  16.0 Predicted value:  16.73154774346129
Actual value:  9.7 Predicted value:  9.779963537030074
Actual value:  14.0 Predicted value:  10.287017178846767
Actual value:  15.5 Predicted value:  15.51848618456846
Actual value:  18.9 Predicted value:  21.126263405128547
Actual value:  12.0 Predicted value:  9.636076815910966
Actual value:  21.4 Predicted value:  23.7124936937626
No: of mislabeled points:  60
Mean Absolute error : 1.3269261137917432
Mean Squared error : 3.72781572306497
Root Mean Squared error : 1.9307552209083807
```

7. Create a neural network for the given ‘houseprice.csv’ to predict the whether price of the house is above or below median value or not

Input:

```
import tensorflow as tf
import keras
import pandas
import sklearn
import matplotlib
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
df = pd.read_csv('housepricedata.csv')
print(df.head())
dataset = df.values
X = dataset[:,0:10]
Y = dataset[:,10]


min_max_scaler = preprocessing.MinMaxScaler()
X_scale = min_max_scaler.fit_transform(X)
print(X_scale)


X_train, X_val_and_test, Y_train, Y_val_and_test = train_test_split(X_scale, Y, test_size=0.3)
X_val, X_test, Y_val, Y_test = train_test_split(X_val_and_test, Y_val_and_test, test_size=0.5)
print(X_train.shape, X_val.shape, X_test.shape, Y_train.shape, Y_val.shape, Y_test.shape)
```

```
model = Sequential([Dense(32, activation='relu', input_shape=(10,)), Dense(32, activation='relu'),Dense(1, activation='sigmoid'),])

model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])

hist = model.fit(X_train, Y_train, batch_size=32, epochs=100, validation_data=(X_val, Y_val))


model.evaluate(X_test, Y_test)[1]


plt.plot(hist.history['loss'])

plt.plot(hist.history['val_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='upper right')

plt.show()
```

Output:

```
   LotArea  OverallQual  OverallCond  ...  Fireplaces  GarageArea  AboveMedianPrice
0     8450            7            5  ...           0         548                 1
1     9600            6            8  ...           1         460                 1
2    11250            7            5  ...           1         608                 1
3     9550            7            5  ...           1         642                 0
4    14260            8            5  ...           1         836                 1

[5 rows x 11 columns]
[[0.0334198  0.66666667 0.5        ... 0.5        0.         0.3864598 ]
 [0.03879502 0.55555556 0.875      ... 0.33333333 0.33333333 0.32440056]
 [0.04650728 0.66666667 0.5        ... 0.33333333 0.33333333 0.42877292]
 ...
 [0.03618687 0.66666667 1.         ... 0.58333333 0.66666667 0.17771509]
 [0.03934189 0.44444444 0.625      ... 0.25       0.         0.16925247]
 [0.04037019 0.44444444 0.625      ... 0.33333333 0.         0.19464034]]
(1022, 10) (219, 10) (219, 10) (1022,) (219,) (219,)
Epoch 1/100
32/32 [==============================] - 0s 5ms/step - loss: 0.6985 - accuracy: 0.4755 - val_loss: 0.6958 - val_accuracy: 0.4475
Epoch 2/100
32/32 [==============================] - 0s 1ms/step - loss: 0.6951 - accuracy: 0.4853 - val_loss: 0.6920 - val_accuracy: 0.4795
Epoch 3/100
32/32 [==============================] - 0s 1ms/step - loss: 0.6919 - accuracy: 0.4922 - val_loss: 0.6886 - val_accuracy: 0.5114
Epoch 4/100
32/32 [==============================] - 0s 1ms/step - loss: 0.6890 - accuracy: 0.5010 - val_loss: 0.6858 - val_accuracy: 0.5342
Epoch 5/100
32/32 [==============================] - 0s 1ms/step - loss: 0.6864 - accuracy: 0.5235 - val_loss: 0.6831 - val_accuracy: 0.5434
Epoch 6/100
32/32 [==============================] - 0s 1ms/step - loss: 0.6837 - accuracy: 0.5352 - val_loss: 0.6804 - val_accuracy: 0.5799
Epoch 7/100
32/32 [==============================] - 0s 1ms/step - loss: 0.6812 - accuracy: 0.5724 - val_loss: 0.6778 - val_accuracy: 0.5982
Epoch 8/100
```

```
Epoch 89/100
32/32 [==============================] - 0s 2ms/step - loss: 0.2953 - accuracy: 0.8855 - val_loss: 0.3059 - val_accuracy: 0.8767
Epoch 90/100
32/32 [==============================] - 0s 1ms/step - loss: 0.2937 - accuracy: 0.8875 - val_loss: 0.3049 - val_accuracy: 0.8767
Epoch 91/100
32/32 [==============================] - 0s 1ms/step - loss: 0.2925 - accuracy: 0.8845 - val_loss: 0.3040 - val_accuracy: 0.8721
Epoch 92/100
32/32 [==============================] - 0s 1ms/step - loss: 0.2905 - accuracy: 0.8904 - val_loss: 0.3065 - val_accuracy: 0.8813
Epoch 93/100
32/32 [==============================] - 0s 2ms/step - loss: 0.2912 - accuracy: 0.8826 - val_loss: 0.3052 - val_accuracy: 0.8858
Epoch 94/100
32/32 [==============================] - 0s 2ms/step - loss: 0.2898 - accuracy: 0.8845 - val_loss: 0.3029 - val_accuracy: 0.8767
Epoch 95/100
32/32 [==============================] - 0s 1ms/step - loss: 0.2881 - accuracy: 0.8836 - val_loss: 0.3019 - val_accuracy: 0.8767
Epoch 96/100
32/32 [==============================] - 0s 1ms/step - loss: 0.2876 - accuracy: 0.8875 - val_loss: 0.3023 - val_accuracy: 0.8767
Epoch 97/100
32/32 [==============================] - 0s 1ms/step - loss: 0.2865 - accuracy: 0.8845 - val_loss: 0.3019 - val_accuracy: 0.8767
Epoch 98/100
32/32 [==============================] - 0s 2ms/step - loss: 0.2860 - accuracy: 0.8894 - val_loss: 0.3011 - val_accuracy: 0.8767
Epoch 99/100
32/32 [==============================] - 0s 2ms/step - loss: 0.2851 - accuracy: 0.8894 - val_loss: 0.3008 - val_accuracy: 0.8767
Epoch 100/100
32/32 [==============================] - 0s 1ms/step - loss: 0.2840 - accuracy: 0.8924 - val_loss: 0.3000 - val_accuracy: 0.8767
7/7 [==============================] - 0s 826us/step - loss: 0.2893 - accuracy: 0.8813
```